QVT-Based Model Transformation
Using XSLT

Dan Li, Xiaoshan Li
Faculty of Science and Technology, University of Macau

Volker Stolz
University of Oslo, Norway
International Institute for Software Technology, UNU
Motivation

- Graphical notation of QVT Relations (QVTR) provides a concise, intuitive way to specify model transformations.
  - But there is NO practical support tool.

- XSLT is a powerful and widely used language with many industrial-strength processors.
  - But programming in XSLT is difficulty during to its low level syntax

- QVTR-XSLT: a practical model transformation framework that combines the power of graphical notation of QVTR and XSLT
OMG’s standards for model-driven development. The core notion includes:

- Metamodelling – defining models
  - MOF: Meta Object Facility
    - Simple class diagrams to define abstract syntax and
    - OCL to define static semantics

- Model Transformation – manipulating models
  - QVT: MOF 2.0 Query, View, and Transformation language
Model Transformation

Define Transformation

Meta-model A

Model A

Conforms To

Transformation model

Apply Transformation

Meta-model B

Model B

Conforms To
QVT Overview

(Taken from the QVT specification)
QVT Relations

- A declarative model transformation language with textual and graphical notation.

- A transformation is specified as a set of relations between model elements of source and target models.

- A relation specifies how two types of object diagrams, called domain patterns, relate to each other.

- Some support tools, but not for graphical notation.
  - Tata Consultancy *ModelMorf*
  - IKV++ medini QVT
QVTR in Graphical Notation

- Provides a concise, intuitive way to specify transformations.

- Graphical specification is a higher-level view that is easier to understand and communicate than the lexical counterpart.

- UML people might expect to continue the graphical tradition of class diagrams and favor a graphical notation

- A picture is worth a thousand words
A Relation in QVTR Graphical Syntax

```
when abs != "true";
where prefix="";
AttributeToColumn(c,t,prefix);AssocEndToFKKey(c,t);
```
Extensible Stylesheet Language for Transformations (XSLT) is one of the W3C standards.

A declarative rule-based programming language for transforming XML documents

Widely used in developing data-intensive applications

An XSLT stylesheet consists of a set of rule templates

Each rule template matches elements in source model, and produces output to the target model.
Why XSLT

- All major CASE tools can export (or import) model as XMI files;
- XSLT is the most common and powerful language for XML transformation;
- XSLT (Xpath) has strong support to complex pattern matching;
- XSLT has many industrial strength implementations, including commercial and open source tools, can also be embedded in Java;
Both QVTR and XSLT are declarative languages. Implementing QVTR as XSLT is done by mapping QVTR expresses to XSLT expresses.

XSLT stylesheet s can be easily executed and integrated into different system environments and platforms, without additional packages and libraries.
<xsl:template match="packagedElement[@xmi:type='uml:Class' and not(@isAbstract='true')]">
  <xsl:variable name="c" select="current()"/>
  <xsl:variable name="cn" select="current()/@name"/>
  <xsl:variable name="prefix" select="''"/>
  <xsl:element name="Table">
    <xsl:attribute name="name" select="$cn"/>
    <xsl:element name="Column">
      <xsl:attribute name="name" select="concat($cn,'_tid')"/>
      <xsl:attribute name="type" select="'integer'"/>
      <xsl:attribute name="key" select="concat($cn,'_pk')"/>
    </xsl:element>
    <xsl:element name="Key">
      <xsl:attribute name="name" select="concat($cn,'_pk')"/>
      <xsl:attribute name="column" select="concat($cn,'_tid')"/>
    </xsl:element>
  </xsl:element>
  <xsl:apply-templates mode="AttributeToColumn" select="$c"/>
  <xsl:with-param name="prefix" select="$prefix"/>
  <xsl:apply-templates/>
  <xsl:apply-templates mode="AssocEndToFKey" select="$c"/>
</xsl:template>
XSLT Cons

- Lower level of abstract
  - verbosity and poor readability of XML

- XSLT programming is different from other program languages

- Requires considerable effort to define complex model transformations directly using XSLT
Get the Best of Both Worlds

- Define the transformation using QVT Relations in graphical notation
- Mapping the transformation into XSLT
- Execute the XSLT program to transform the source model to target model
QVTR-XSLT Tool

- A Graphical Editor
  - support design of QVTR transformations in graphical notation;
  - save the QVTR transformation models as XML files.

- A Code Generator
  - reads in the transformation model, generates corresponding XSLT stylesheets.
QVTR Graphical Editor

- Built on top of *MagicDraw UML*, a popular UML CASE tool;
- A UML profile to define QVTR transformation models;
- A toolbar to edit QVTR diagrams;
- A set of OCL rules to validate the transformation models.
UML Profile for QVTR Model

- **stereotype**
  - QVTRModel
    - [Model]
    - `+currentTransformation : Transformation`
- **stereotype**
  - MetaModel
    - [Package]
- **stereotype**
  - Relation
    - [Package]
    - `+isTopLevel : Boolean`
    - `+name : String`
    - `+priority : Integer`
- **stereotype**
  - Domain
    - [InstanceSpecification]
    - `+kind : DomainKind`
- **stereotype**
  - Transformation
    - [Package]
    - `+source : MetaModel`
    - `+sourceName : String`
    - `+sourceIsCheckable : Boolean`
    - `+sourceIsEnforceable : Boolean`
    - `+target : MetaModel`
    - `+targetName : String`
    - `+targetIsCheckable : Boolean`
    - `+targetIsEnforceable : Boolean`
    - `+isInPlace : Boolean`
    - `+sourceKey : String`
    - `+targetKey : String`
    - `+makeTrace : Boolean`
    - `+output : TransformationOutput`
    - `+includeFile : String`
    - `+parameterFile : String`
    - `+linkAsSubElement : Boolean`
    - `+extends : String`
- **stereotype**
  - Function
    - [FunctionBehavior]
    - `+notForMatch : Boolean`
    - `+notForCreate : Boolean`
    - `+position : Integer`
    - `+member : String`
- **stereotype**
  - Key
    - [OpaqueBehavior]
    - `+when : PrimitiveDomain`
    - [InstanceSpecification]
    - `+sNumber : Integer`
- **stereotype**
  - When
    - [Constraint]
- **stereotype**
  - Where
    - [Constraint]

- **enumeration**
  - DomainKind
    - `source`
    - `target`
- **enumeration**
  - TransformationOutput
    - `xml`
    - `html`
QVTR Transformation Model

Consist of:
- **MetaModels**
  - classes, associations
  - *class diagrams*
- **Transformations**
- **Relations**
  - Domains
  - Objects and Links
  - When clause
  - Where clause
  - QVTR *diagram*
- **Functions**
Toolbar for QVTR Diagram

```java
PrimitiveAttributeToColumn
{where=cn = if (prefix = '') then an else prefix + ' ' + an endif;
sqltype = PrimitiveTypToSqlType(pn);}
```

- **c**: Class
  - name = "an"

- **prefix**: String

- **t**: Table
  - name = "cn"
  - type = "sqltype"

- **cl**: Column
  - name = "cn"
  - type = "sqltype"
QVTR Transformation Model Validation

[Image of validation tool interface with the following settings:
- Validation Suite: QVTR.Validate [QVTR]
- Validate For: Whole Project
- Minimal Severity: >=debug
- Exclude elements from read-only modules]

[Buttons: Validate, Cancel, Help]
Overall Interface of the QVTR Editor
The generator itself is an XSLT stylesheet;

It reads in the XML file saved from the transformation model, analyzes the model’s structure, parses the OCL expressions, and generates an XSLT stylesheet that represents the QVTR transformation.
### Mapping Transformation to Stylesheet

<table>
<thead>
<tr>
<th>QVTR</th>
<th>XSLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation</td>
<td>Stylesheet</td>
</tr>
<tr>
<td>Relation</td>
<td>Rule template</td>
</tr>
<tr>
<td>Primitive domain</td>
<td>Template parameter</td>
</tr>
<tr>
<td>Function</td>
<td>Function</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>OCL expression</td>
<td>XPath expression</td>
</tr>
</tbody>
</table>
Mapping Relation to Rule Template

**Relation**
- Source domain pattern
- When clause
- Source domain pattern
- Where clause
- Target domain pattern
- Relation calls

**Rule Template**
- Match expression
- Variable declarations
- Construction instructions
- Template calls
Mapping OCL to XPath

**OCL**:  
```
dn.outgoing->
    select(name='else')->size()=1
```

**XPath**:  
```
count(my:xmiXMIrefs(current()/
    @outgoing)[@name='else'])=1
```
Comparison - UML to RDBMS Example

- XSLT generated by the tool: 130 lines of code
- QVT relations in textual: 120 lines
- QVT operational: 100 lines
- QVT core: 400 lines
- MT: 140 lines
- medini QVT: 240 lines
Tool Features

- Unidirectional transformation;
- Single source model, target model creating;
- Complex pattern matching of object templates, property templates, collection templates, and *not* templates;
- OCL expression referenced source domain pattern elements;
- In-place transformation;
- Transformation parameters;
- Transformation extensions (inheritance);
- Execution trace output;
Potential Tool Users

- Model Community
  - A practical QVTR-compliance tool with graphical syntax support;

- XML Community
  - A higher-level XSLT generator with user-friendly IDE
Case Studies

- Model-to-Model
  - UML to RDBMS transformation
  - UML Activity Diagrams to CSP transformation

- Model-to-Text (Html)
  - CSP to Html transformation
  - RDBMS to SQL transformation

- In-Place
  - Multiplicity to OCL transformation
  - Small-step refinement
- Thank You!